A Git Source Repository Analysis Tool Based on a Novel Branch-oriented Approach

HaeJun Lee
School of Information
and Communication Engineering
Sungkyunkwan University
Email: vhxmz@skku.edu

Bon-Keun Seo
Computer Science Department
Korea Advanced Institute of
Science and Technology
Email: bkseo@calab.kaist.ac.kr

Euiseong Seo
School of Information
and Communication Engineering
Sungkyunkwan University
Email: euiseong@skku.edu

***Abstract*—Code repositories of the version control systems have been used as important raw material in lots of developer behavior studies. In such research, commits are usually considered as a unit of development phases, even though they contain limited information over development process. Unlike former source code management systems, Git provides branching, which are the unit of concurrent and independant development activities in a whole project. Branches have significant advantages over commits from the viewpoint of the developer behavior researchers. By exploiting the characteristics of branching, we propose a Git branch analysis tool. The proposed tool automatically extracts branching data from Git repositories and performs statistical and graphical analysis on them. Through a few case studies using a number of popular OSS project repositories, we showed that the proposed branch-oriented analyzer is able to perform more insightful analysis than the existing commit-based analyzers.**

I. Introduction

The researchers who study the development process of open source software (OSS) projects require raw material to investigate the development history. They have been looking for the sources of the necessary information, and found them from the source code repositories, bug DBs, developer mailing lists, and forums on the web [1]–[4], which are available to public through Internet.

Obtaining standardized and trustful data from bug DBs or web logs is, however, technically challenging since such data are usually unstructured. On the contrary, obtainable data from the source code repositories, such as the number of code lines, the number of commits done and so on, are usually well-formatted and structured. In addition, they record every change of all files in a project. This allows researchers to reconstruct the development history along the long time evolution of the project. Therefore, the source code repositories have been drawing enormous interest from lots of researchers [5], [6].

Due to the great success of a few representative OSS projects including Linux and Apache, tons of OSS projects have been created for the last two decades, and their source repositories naturally have been publicly open. Thus the project hosting services that collect the repositories and serve them to public are being widely used these days. For example, *SourceForge.net* and *Github.com* are hosting more than 100,000 projects and roughly 3.4 million users working on 6 million repositories as of 2007 [4].

These hosting services are the favorite sources of data for the researchers of developer behavior and development process because they are easily accesible rich data. For the same reason, Sowe et al. suggested a repository of repositories [7] so that the data regarding the development process and developer behavior become easily obtainable for the researchers.

Naturally, lots of repository analysis tools, such as *CVSAnalY*, have been developed to extract various data from the public OSS repositories [1], [8], [9]. Nevertheless, the commit logs are the focus of these tools since they are the building blocks of the repository. Not only the earlier study of Linux [5] and Apache [3], but also the recent researche results [6], [10] use the commits as a unit of a development task.

Although investigation using commit history of a project provides useful information of the development flow, the commit history is lack of the relationships among modular development tasks in a project since the commits for different components in a project are linearly and sequentially done in a chronical order.

By the nature of the OSS development model, developers are inherently spreaded over the world, and they work on what they are interested at their convenient time. This nature leads the OSS development process to consist of complex concurrent tasks. However, as stated, the commit history is not able to depict this parallel development records. Therefore, in this research, we aim at proposing a tool that is able to grasp the concurrent development activities by exploiting the branch history, which is provided by the Git source code manager.

Git is currently one of the most popular source code management tools for the huge number of OSS projects. Consequently, the researchers are able to obtain plentiful data sources for the development behavior analysis research when they have an adequate tool that utilizes the Git repositories. The proposed tool, Git Branch Analyzer, retrieves the branching and merging history from a Git repository and reconstruct the development activities based on the retrieved data. Also, it extracts diverse and useful information from the reconstructed development activity history.

The rest of the paper is organized as follows. Section II introduces the Git source code management system and proposes the branch-oriented developer bahavior analysis tool.

This research was supported by Next-Generation Information Computing Development Program (2012-0006423) and Basic Science Research Program (2012R1A1A2A1A10038823) through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology.

978-1-4799-0604-8/13/$31.00 ©2013 IEEE

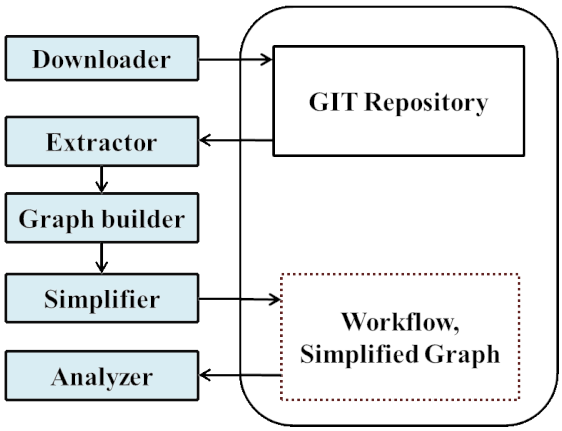


Fig. 1: Design overview of Git branch analyzer.

In Section III, we prove the effectiveness of the proposed tool by demonstrating three use cases of the proposed tool using the data set collected from *GitHub.com*. Finally, we introduce the future research direction and conclude this paper in Section IV.

II. Git Branch Analyzer

This section introduces a typical development workflow using the Git system, and then illustrates the design and implementation of the Git Branch Analyzer.

A. Git Workflow

Knowing how the developers use a Git repository is the first step to determine what to extract from them.

Few commits are enough to manage changes in the source code when the number of changed code is small. However, a branch must be forked to encapsulate a development task when significant portion of code must be modified by the task. There may be a large number of commits in the newly forked branch. However, such commits do not affect the code base of the other branches. Once the development module for a branch is completed, the branch will be merged into the parent branch.

Git provides strong supports for the branch and merge operations. One can make a new branch and work on it until he thinks it is stable enough to be merged . It is completely up to the user that the branch will be merged to the branch it has forked from or the other depending on the characteristics of the changes. There is no limitation on the number of branches that concurrently exist at a certain time point. Developers can create cross-references over the development tasks while isolating each workflow. This extremely flexible branch structure enables developers not only to maximize their productivity, but also to efficiently manage the simultaneous development activities.

B. Design Goal

Branches, which have been ignored by the previous studies, are the key concern of the aimed tool of this research, and differentiate it from the other existing repository analysis tools. Git Branch Analyzer aims at not only recognizing branches and collecting them, but also analyzing them statistically and graphically.

First, it needs to extract the details of commit logs from the repository. The details are as follows: the code changes in the commit, the time of the commit, the committer, the author, and so on. Also, we can make use of commit tags to visualize the changes in the more informative way.

Second, it must be able to reconstruct branches from the repository logs. Consulting the commit hierarchy of a repository, it builds a sequence of commits that constitute each branch.

Last but not least, it should be able to build graphs illustrating the relationships among branches. The generated graphs will help researchers to understand the information about when, where, and how many branches are created and merged as time flows. Also these graphs will reveal how a development unit interacts with the other development tasks. In addition, the proposed tool should provide methods to analyze the graphs chronologically.

C. Implementation Details

The Git branch analyzer is written in Python. The structural overview is depicted in Figure 1. It uses a Git repository as the data source, which is cloned via `Git clone` command. The analyzer inspects commit logs, identifies workflows, and builds a branching graph as follows.

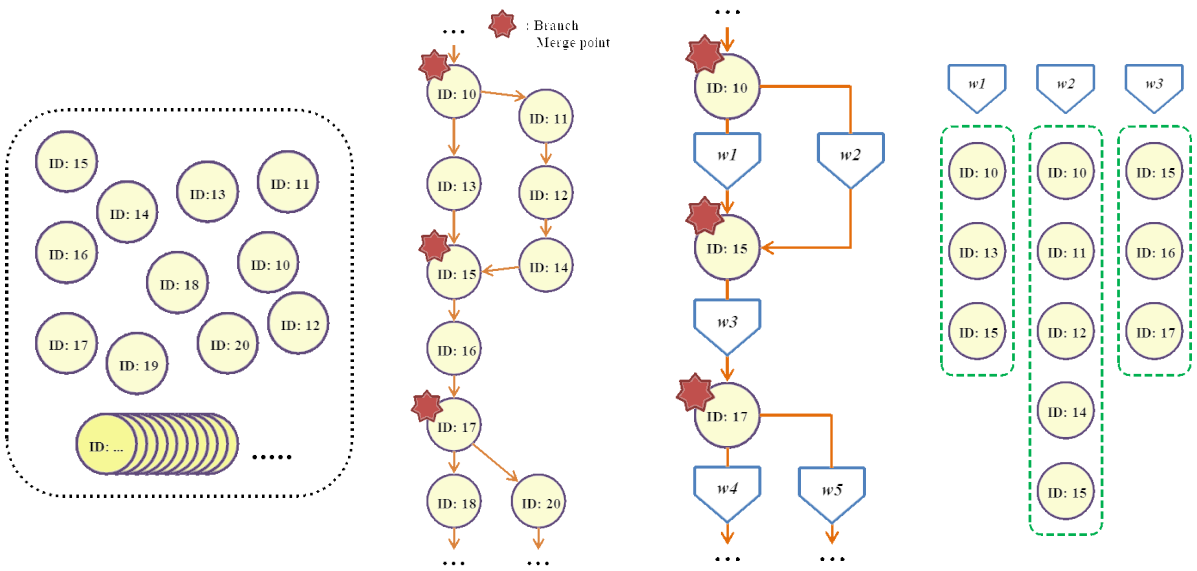
Download : The target Git repository is downloaded to local storage by using the `Git clone` command. Since the tool uses the standard Git clone command, it can download from any available Git repositories including the repositories provided by Github.com.

Extract : It reads the commit logs in the cloned repository as shown in Figure 2a. The details such as commit hashes, parent commit hashes, commit time stamps, committers, author time stamps, authors, tree hashes, and commit messages are fetched and sorted in the chronological order. These data will be the basis for constructing the branching graph.

Structurize : The commit graph is built from the extracted commit logs as depicted in Figure 2b. Building nodes from the commit logs is the first step of this stage. In the graph, each node represents each commit log item, and the links between them relate the corresponding commit sequences. The graphs gradually grow by attaching additional commit nodes to their parent commit nodes, which are identifiable through the parent commit IDs in a commit log. The resulting graph becomes a directed graph, which has one starting node and a number of ending nodes that have no child nodes.

Simplify : Most of the nodes in the commit graph have one parent and one child. Only small fraction of nodes has multiple parents or children, which are branching or merging point. Successful and aged OSS projects tend to show huge number of commits and significantly complicated graph structure. To expedite the forthcoming scan and analysis tasks over the constructed graph, the tool simplifies the graph.

The nodes that has only one parent and one child are merged to form a unit workflow, which consists of a branching point, a merging point and multiple intermediate commits in between the branching and merging points as shown in



(a) Raw commit data. (b) Commit graph. (c) Workflow graph. (d) Workflow data.

Fig. 2: The steps to build workflows from Git repository.

TABLE I: Statistical data of selected OSS projects

Project	Number of commits	Number of committer	LOC of last commit	Number of workflow
Linux kernel	348,809	412	16,414,157	54,812
Android kernel	300,317	371	15,497,543	46,042
backbone	2,106	198	67,608	1,289
bootstrap	3,569	256	66,026	1,740
CodeIgniter	6,384	307	100,294	2,874
chosen	393	56	7,086	202
d3	2,507	54	45,509	994
foundation	2,977	186	39,720	1,881
game-of-life	413	6	845,632	114
Gitignore	567	170	2,029	375
homebrew	19,584	51	91,406	35
html5-boilerplate	1,231	137	3,108	509
impress.js	206	24	2,288	87
jquery	5,168	81	35,891	653
jQuery-File-Upload	514	26	7,080	70
jquery-ui	5,496	94	204,552	440
node	8,350	20	2,373,667	675
oh-my-zsh	1,620	421	15,223	1,204
phonegap-plugins	1,458	209	947,748	1,033
rails	37,029	1,568	315,981	10,879
symfony	13,603	615	312,047	7,676
three	5,525	182	786,046	2,577

Figure 2d. The branching and merging points in the commit graph are marked as starred points as in Figure 2b, and remain in the simplified graph as shown in Figure 2c. All other nodes are removed in the simplified graph. A workflow becomes an edge in the simplified graph, and it represents the creation and completion of a unit development activity or task.

Analyze : Finally, the graph and workflows are analyzed to generate useful statistical information. The data studied in the existing research, such as the number of commits, committers, commit time stamps, and LoC (lines of code), can be extracted easily. In addition to that, workflows are easily browsed via simplified graph. Diverse graph algorithms can be applied to the simplified graph or the original commit graph to investigate diverse properties of the development process and developer behavior. Some examples of the powerful analysis functions of the proposed tool are demonstrated in Section III.

III. Case Studies

In order to demonstrate the effectiveness and superiority of the proposed tool, this section introduces a few case studies that utilize the tool to investigate the characteristics of the development process of the representative OSS projects.

A. Statistical Analysis

How many LoC are there in an OSS project? How many commits are there in a project? To show the capability of the proposed tool in answering these questions, we analyzed the Linux kernel repository as well as the top 20 popular project repositories of Github.com. Table I lists the repositories used in the case studies. The popularity of the Github repositories was determined by the number of forks and stars a repository earns.

The categories of the collected data are as follows: the size of LoC of the last commit, the number of commits, and the number of committers. These are only a small set of information that can be obtained by the proposed tool, and can be easily obtainable also by the existing analyzers. The comitters who have an identical name but have different e-mail addresses are merged into one since a developer can have multiple e-mail addresses.

As shown in the table, the basic information about a repository were successfully extracted by the proposed tool. Other fragmentary data can be also easily acquired by the minor modification of the tool.

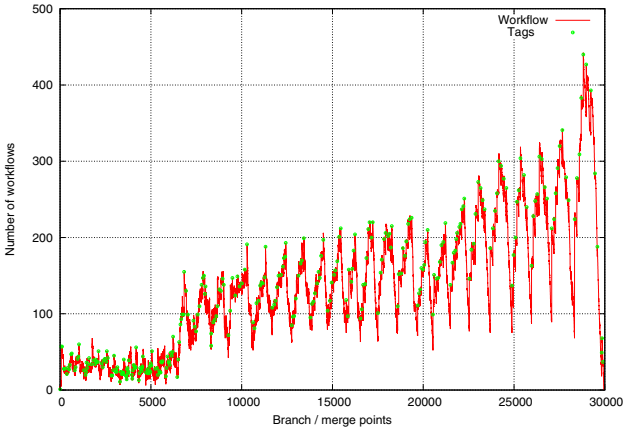


Fig. 3: Concurrent workflows in the Linux kernel project.

B. Graphical Branch Analysis

Because a workflow is the set of commits between a branch and a merge, it represents a unit of development activities that a developer does for implementation of a new feature, repair of a bug and so on [11]. By analyzing the workflows and their graphical relations, it is possible to understand more deeply about how developers work in cooperation with the others. Our branch analyzer makes it possible to analyze the workflows.

As an example, we gathered the number of concurrent workflows at each branch and merge points from the latest Linux kernel repository. For that purpose, we extracted the first and last commit time of each workflow, and then counted the number of workflows which are active at each time point when a branching or merging happens.

The result is depicted in Figure 3. We marked the points where each stable or release candidate version of the Linux kernel was released with green dots. The result revealed some interesting facts, which have been unknown so far.

First, the number of concurrent works grows as the Linux kernel becomes mature. It can be interpreted as to; the maturity of a successfull project provokes more and more development activities with time.

Second, the number of concurrent workflows fluctuates as time goes on. This property is strongly related to the development cycle of the Linux kernel. It can be explained as follows: a lot of branches are created after a stable release, and merged before the next stable release.

Third, branches are used conservatively at the first time, but it is used more proactively as time goes by.

From this case study, it was revealed that the graphical branch analysis leads to the detailed and informative descriptions on how developers cooperate and how OSS projects evolve over time.

IV. Conclusion and Future Work

We designed and implemented a Git branch analyzer that extracts statistical and graphical data from Git repositories. By using a branch as a unit of a development task within a project, the proposed tool enables comprehensive understanding of developers' cooperation patterns. In addition, the proposed analyzer automates retrieving data from repositories accomodated in huge public domain OSS project hosting services so that it helps researchers to utilize the massive-scale data for their research. In this research, we proved the effectiveness of our approach by introducing a few case studies with the data obtained from *GitHub.com*.

For the next step, we are planning to extensively study developers' behavior in a large set of OSS projects. For example, we can differentiate successful projects and failed (or abandoned) projects by applying data mining techniques to statistical or graphical data that can be derived from the branching patterns by using the proposed tool. Also, we can identify the characteristics of branches for various kinds of tasks, such as fixing bugs, adding new features, porting to other hardware, etc.

References

[1] G. Robles, J. M. González-Barahona, R. A. Ghosh, and J. Carlos, "GlueTheos: Automating the retrieval and analysis of data from publicly available software repositories," in *Proceedings of the International Workshop on Mining Software Repositories*, 2004, pp. 28–31.

[2] G. Robles, S. Koch, and J. M. González-Barahona, "Remote analysis and measurement of libre software systems by means of the CVSAnalY tool," in *Proceedings of the Second ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS'04)*, 2004, pp. 51–55.

[3] A. Mockus, R. T. Fielding, and J. Herbsleb, "A case study of open source software development: The Apache server," in *Proceedings of the 22nd International Conference on Software Engineering*, 2000, pp. 263–272.

[4] S. Christley and G. Madey, "Analysis of activity in the open source software development community," in *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, 2007, pp. 166b–166b.

[5] M. W. Godfrey and Q. Tu, "Evolution in open source software: A case study," in *Proceedings of International Conference on Software Maintenance*, 2000, pp. 131–142.

[6] T. Mens and M. Goeminne, "Analysing the evolution of social aspects of open source software ecosystems," in *Proceedings of the 3rd International Workshop on Software Ecosystems (IWSECO)*, 2011, pp. 1–14.

[7] S. Sowe, L. Angelis, I. Stamelos, and Y. Manolopoulos, "Using repository of repositories (RoRs) to study the growth of F/OSS projects: A meta-analysis research approach," in *Open Source Development, Adoption and Innovation*, ser. IFIP–The International Federation for Information Processing. Springer US, 2007, vol. 234, pp. 147–160.

[8] D. German and A. Mockus, "Automating the measurement of open source projects," in *Proceedings of the 3rd Workshop on Open Source Software Engineering*, 2003, pp. 63–67.

[9] B. Massey and K. Packard, "Regurgitate: Using GIT for F/LOSS data collection," in *Proceedings of the 1st International Workshop on Public Data about Software Development*, Milan, Italy, 2006.

[10] G. Gousios and D. Spinellis, "GHTorrent: GitHub's data from a firehose," in *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR)*, 2012, pp. 12–21.

[11] J. Loeliger and M. McCullough, *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*, 2nd ed. O'Reilly, 2012.